(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 December 2003 (04.12.2003)

PCT

(10) International Publication Number
WO 03/101037 A1

(51) International Patent Classification[7]: H04L 9/00

(21) International Application Number: PCT/US03/16445

(22) International Filing Date: 22 May 2003 (22.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/154,070 23 May 2002 (23.05.2002) US

(71) Applicant: SYMANTEC CORPORATION [US/US]; 20330 Stevens Creek Boulevard, Cupertino, CA 95014 (US).

(72) Inventor: SZOR, Peter; 937 12th Street, #205, Santa Monica, CA 90403 (US).

(74) Agents: HOFFMAN, Brian M. et al.; Fenwick & West LLP, Silicon Valley Center, 801 California Street, Mountain View, CA 94041 (US).

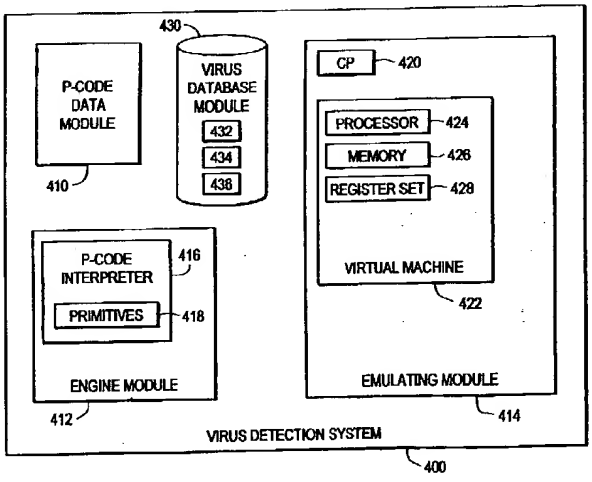
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METAMORPHIC COMPUTER VIRUS DETECTION



(57) Abstract: The executions of computer viruses are analyzed to develop register signatures for the viruses. The register signatures specify the set of outputs the viruses produce when executed with a given set of inputs. A virus detection system (VDS) (400) holds a database (430) of the register signatures. The VDS (400) a file that might contain a computer virus and identifies potential entry points in the file. The VDS (400) uses a virtual machine (422) having an initial state to emulate a relatively small number of instructions at each entry point. While emulating each potential entry point, the VDS builds a register table that tracks the state of a subset of the virtual registers (428). Once the VDS (400) reaches an emulation breakpoint, it analyzes the register table in view of the register signatures to determine whether the file contains a virus.

WO 03/101037 A1

METAMORPHIC COMPUTER VIRUS DETECTION

INVENTOR
PETER SZOR

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

[0001] This invention pertains in general to detecting computer viruses and in particular to detecting metamorphic computer viruses.

BACKGROUND ART

[0002] Modern computer systems are under constant threat of attack from computer viruses and other malicious code. Viruses often spread through the traditional route: a computer user inserts a disk or other medium infected with a virus into a computer system. The virus infects the computer system when data on the disk are accessed.

[0003] Viruses also spread through new routes. A greater number of computer systems are connected to the Internet and other communications networks than ever before. These networks allow a networked computer to access a wide range of programs and data, but also provide a multitude of new avenues by which a computer virus can infect the computer. For example, a virus can be downloaded to a computer as an executable program, as an email attachment, as malicious code on a web page, etc. Accordingly, it is common practice to install anti-virus software on computer systems in order to detect the presence of viruses.

[0004] Simple computer viruses work by copying exact duplicates of themselves to each executable program file they infect. When an infected program is executed, the simple virus gains control of the computer system and attempts to infect other files. If the virus locates a

target executable file for infection, it copies itself byte-for-byte to the target executable file. Because this type of virus replicates an identical copy of itself each time it infects a new file, the anti-virus software can detect the virus quite easily by scanning the file for a specific string of bytes (i.e. a "signature") characteristic of the virus.

[0005] The designers of computer viruses are constantly evolving new techniques for eluding the anti-virus software. Encrypted viruses are examples of one such technique. Encrypted viruses include a decryption routine (also known as a "decryption loop") and an encrypted viral body. When a file infected with an encrypted virus executes, the decryption routine gains control of the computer and decrypts the encrypted viral body. The decryption routine then transfers control to the decrypted viral body, which is capable of spreading the virus. The virus spreads by copying the identical decryption routine and the encrypted viral body to the target executable file. Although the viral body is encrypted and thus hidden from view, anti-virus software can detect these viruses by searching for a signature in the unchanging decryption routine.

[0006] A polymorphic encrypted virus ("polymorphic virus") includes a decryption routine and an encrypted viral body. The viral body includes a static portion and a machine-code generator often referred to as a "mutation engine." The operation of a polymorphic virus is similar to the operation of an encrypted virus, except that the polymorphic virus generates a new decryption routine each time it infects a file. Many polymorphic viruses use decryption routines that are functionally the same for all infected files, but have different sequences of instructions.

[0007] These multifarious mutations allow each decryption routine to have a different signature. Therefore, anti-virus software cannot detect polymorphic viruses by simply searching for a signature from a decryption routine. Instead, the software loads a possibly-

infected program into a software-based CPU emulator acting as a simulated virtual computer. The program is allowed to execute freely within this virtual computer. If the program does in fact contain a polymorphic virus, the decryption routine is allowed to decrypt the viral body. The anti-virus software detects the virus by searching through the virtual memory of the virtual computer for a signature from the decrypted viral body.

[0008] Virus creators have produced "metamorphic" viruses in order to defeat the signature scanning-based detection techniques described above. Metamorphic viruses are not necessarily encrypted, but vary the instructions in the viral body with each infection. Common techniques utilized by metamorphic viruses include inserting jump instructions after every instruction in the viral body, using varying sets of equivalent instructions to perform certain functions, inserting no-operation instructions at random places in the code, etc. One sophisticated metamorphic virus actually disassembles a target host file, inserts the virus code amongst the disassembled host code, and then reassembles the host file.

[0009] Accordingly, anti-virus software often cannot detect metamorphic viruses because there are no static signatures for which to search. Therefore, there is a need in the art for a technique that can reliably detect the presence of metamorphic viruses.

DISCLOSURE OF INVENTION

[0010] The above need is met by a virus detection system (VDS) (400) that uses register signatures to detect metamorphic and other types of viruses. The VDS (400) preferably includes a data module (410), an engine module (412), an emulating module (412), and a virus database module (430). The operation of the VDS (400) is preferably driven by P-code

instructions stored in the data module (410). The engine module (412) includes a P-code interpreter (416) for interpreting the P-code and controlling the VDS (400) in response.

[0011] The emulating module (412) is controlled by the engine module (412) and executes computer program instructions in a virtual machine (422) having a virtual processor (424), virtual registers (428), and a virtual memory (426). The virtual machine (422) executes the instructions in isolation from the actual hardware and software on the computer system (200) so that a virus in the virtual machine cannot infect files on the computer system.

[0012] The virus database module (430) preferably stores register signatures utilized by the VDS (400) to detect the presence of viruses. A register signature describes the values a virus stores in the registers while executing, and the order in which the virus stores the values. The register signatures are effective because a computer program, e.g., a virus, will produce a given set of outputs, e.g., the register values, when executed with a given set of inputs. This statement generally holds true even for code-altering metamorphic viruses.

[0013] The P-code instructions in the data module (410) preferably include a file selection module (510) for filtering the files on the computer system (200) to identify potential virus host files. An emulation control module (520) identifies and selects potential virus entry points in the selected file, and causes the emulating module (412) to emulate instructions at the entry points until reaching breakpoint conditions. While the emulating module (412) emulates the instructions, a table builder module (522) builds a table (600) tracking the values of the virtual registers (428) at each step. Preferably, the emulating module (412) emulates relatively few (e.g., 50-500) instructions at each of the selected entry points.

[0014] Once the emulation reaches a breakpoint or otherwise stops, a virus reporting module (526) analyzes the table to determine whether it contains any of the register signatures

stored in the virus database module (430). If a register signature matches, then the selected file probably contains a virus.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] FIG. 1 is a high-level block diagram illustrating a potential virus host file 100 having multiple entry points;

[0016] FIG. 2 is a high-level block diagram illustrating a computer system 200 for executing a virus detection system (VDS) 400;

[0017] FIG. 3 is a flowchart illustrating steps performed by a typical metamorphic virus when infecting a host file 100;

[0018] FIG. 4 is a high-level block diagram illustrating modules of the VDS 400 according to an embodiment of the present invention;

[0019] FIG. 5 is a high-level block diagram illustrating a more detailed view of the P-code data module 410 in the VDS 400;

[0020] FIG. 6 illustrates a register table 600 according to an exemplary embodiment of the present invention; and

[0021] FIG. 7 is a flowchart illustrating the operation of the VDS 400 to detect the presence of a virus according to an embodiment of the present invention.

[0022] The figures depict an embodiment of the present invention for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention described herein.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0023] In order to accomplish the mischief for which they are designed, computer viruses must gain control of a computer's central processing unit (CPU). Viruses typically gain this control by attaching themselves to an executable file (the "host file") and modifying the executable image of the host file to pass control of the CPU to the viral code. The virus conceals its presence by executing its own instructions and then calling the original instructions in order to pass control back to the host file. As used herein, the term "virus" also includes other forms of malicious code such as "worms" and "Trojan horses" that can infiltrate and infect a computer system.

[0024] Viruses use different techniques to infect the host file. For example, a simple virus always inserts the same viral body into the file. An encrypted virus infects a file by inserting an unchanging decryption routine and an encrypted viral body into the host file. A polymorphic encrypted virus (a "polymorphic virus") is similar to an encrypted virus, except that a polymorphic virus generates a new decryption routine each time it infects a file. A metamorphic virus is not necessarily encrypted, but it reorders the instructions in the viral body into a functionally equivalent, but different, virus each time it infects a file. A virus may use a combination of the techniques described above. Although an advantage of the present invention is that it can reliably detect metamorphic viruses, it should be understood that the present invention can be used to detect any type of virus.

[0025] A virus typically infects a host file by attaching or altering code at or near an entry point of the file. An "entry point" is any instruction or instructions in the file, a pointer to an instruction or instructions, or other data in the file that a virus can modify to gain control of the computer system at which the file is located. An entry point is typically identified by an offset

from some arbitrary point in the file. Certain entry points are located at the beginning of a file or region and, thus, are always invoked when the file or region is executed. For example, an entry point can be the first instruction executed when the host file is executed or a function within the file is called. Other entry points may consist of single instructions deep within the file that can be modified by a virus. For example, the entry point can be a CALL or JMP instruction that is modified to invoke viral code. Once a virus seizes control of the computer system through the entry point, the virus typically infects other files on the system or on other systems networked with the host system.

[0026] FIG. 1 is a high-level block diagram of a potential virus host file 100 having multiple entry points that can be infected by a virus as described above. In the example illustrated by FIG. 1, the executable file is a Win32 portable executable (PE) file intended for use with a MICROSOFT WINDOWS-based operating system (OS), such as WINDOWS ME, WINDOWS NT, WINDOWS 2000, or WINDOWS XP. Typically, the illustrated file 100 is of the type .EXE, indicating that the file is an executable file, or .DLL, indicating that the file is a dynamic link library (DLL). However, the present invention can be used with any file, and is not limited to only the type of file illustrated in FIG. 1. APPLE MACINTOSH files, for example, share many similarities with Win32 files, and the present invention is equally applicable to such files.

[0027] The file 100 is divided into sections containing either code or data and aligned along four kilobyte (KB) boundaries. The MS-DOS section 102 contains the MS-DOS header 102 and is marked by the characters "MZ." This section 102 contains a small executable program 103 designed to display an error message if the executable file is run in an unsupported OS (e.g., MS-DOS). This program 103 is an entry point for the file 100. The

MS-DOS section 102 also contains a field 104 holding the relative offset to the start 108 of the PE section 106. This field 104 is another entry point for the file 100.

[0028] The PE section 106 is marked by the characters "PE" and holds a data structure 110 containing basic information about the file 100. The data structure 110 holds many data fields describing various aspects of the file 100.

[0029] The next section 112 holds the section table 114. The section table 114 contains information about each section in the file 100, including the section's type, size, and location in the file 100. For example, entries in the section table 114 indicate whether a section holds code or data, and whether the section is readable, writeable, and/or executable. Each entry in the section table 114 describes a section that may have multiple, one, or no entry points.

[0030] The text section 116 holds general-purpose code produced by the compiler or assembler. The data section 118 holds global and static variables that are initialized at compile time.

[0031] The export section 120 contains an export table 122 that identifies functions exported by the file 100 for use by other programs. An EXE file might not export any functions but DLL files typically export some functions. The export table 122 holds the function names, entry point addresses, and export ordinal values for the exported functions. The entry point addresses typically point to other sections in the file 100. Each exported function listed in the export table 122 is an entry point into the file 100.

[0032] The import section 124 has an import table 126 that identifies functions that are imported by the file 100. Each entry in the import table 126 identifies the external DLL and the imported function by name. When code in the text section 116 calls a function in another module, such as an external DLL file, the call instruction transfers control to a JMP instruction

also in the text section 116. The JMP instruction, in turn, directs the call to a location within the import table 126. Both the JMP instruction and the entries in the import table 126 represent entry points into the file 100.

[0033] FIG. 2 is a high-level block diagram of a computer system 200 for storing and executing the host file 100 and a virus detection system (VDS) 400. Illustrated are at least one processor 202 coupled to a bus 204. Also coupled to the bus 204 are a memory 206, a storage device 208, a keyboard 210, a graphics adapter 212, a pointing device 214, and a network adapter 216. A display 218 is coupled to the graphics adapter 212.

[0034] The processor 202 may be any general-purpose processor such as an INTEL x86, SUN MICROSYSTEMS SPARC, or POWERPC compatible-CPU. As is known in the art, the processor 202 has multiple registers that are used to hold instructions, data, addresses, and other information. The storage device 208 may be any device capable of holding data, like a hard drive, compact disk read-only memory (CD-ROM), DVD, or a solid-state memory device. As is known in the art, the storage device 208 typically holds multiple files of different types, including potential virus host files like that described by FIG. 1. The memory 206 holds instructions and data used by the processor 202. The pointing device 214 may be a mouse, track ball, or other type of pointing device, and is used in combination with the keyboard 210 to input data into the computer system 200. The graphics adapter 212 displays images and other information on the display 218. The network adapter 216 couples the computer system 200 to a local or wide area network.

[0035] As is known in the art, the computer system 200 is adapted to execute computer program modules for providing functionality described herein. As used herein, the term "module" refers to computer program logic and/or any hardware or circuitry utilized to provide the specified functionality. Thus, a module can be implemented in hardware, firmware, and/or

software. Preferably, program modules providing the functionality of the VDS 400 are stored on the storage device 208.

[0036] FIG. 3 is a flowchart illustrating steps performed by a typical metamorphic virus when infecting the host file. The illustrated steps are merely an example of a viral infection and are not representative of any particular virus. Initially, the virus executes 310 on the computer system 200. The virus may execute, for example, when the computer system 200 executes or calls a function in a previously infected file.

[0037] The virus identifies 312 a new host file to infect. For example, the virus may search for files matching the string "*.EXE" to identify new host file candidates. The virus then filters the candidate files to identify a particular new host file 100. The files may be filtered based on the files' sizes, names, whether the files are already infected by the virus, etc.

[0038] The virus inserts 314 its viral code into the new host file. For example, the virus can append the viral body to the slack space at the end of a section or put the viral body within an entirely new section. During this step, the virus uses one or more techniques to modify its current viral code to produce new viral code, which it then inserts into the file 100. The techniques the virus can use to modify its code include inserting and/or removing jump ("JMP") and no-operation ("NOP") instructions, substituting varying sets of equivalent instructions for performing certain tasks, such as register subtractions or zeroing, and/or substituting the registers utilized by certain instructions.

[0039] The virus typically modifies 316 the section table 114 in the host file 100 to account for the added viral code. For example, the virus may change the size entry in the section table 114 to account for the added viral code. Likewise, the virus may add entries for new sections added by the virus. If necessary, the virus may mark an infected section as

executable and/or place a value in a little-used field to discreetly mark the file as infected and prevent the virus from re-infecting the file 100.

[0040] In addition, the virus typically also alters 318 an entry point of the host file 100 to call the viral code. The virus may accomplish this step by, for example, overwriting the value in the field 104 holding the relative offset to the start 108 of the PE section 106 with the relative offset to virus code stored elsewhere in the file. Alternatively, the virus can modify entries in the export table 122 to point to sections of virus code instead of the exported functions. A virus can also modify the destination of an existing JMP or CALL instruction anywhere in the file to point to the location of viral code elsewhere in the file, effectively turning the modified instruction into a new entry point for the virus.

[0041] A particularly sophisticated virus can use very complex techniques for infecting the host file that differ in some respects from the ones described above. For example, one sophisticated virus includes a disassembly engine in its viral body and uses it to disassemble the host file into its smallest elements. Then, the virus infects the dissembled host file by moving code blocks to clear space for the viral code, inserting its modified viral body, regenerating code and data references, including relocation information, and then rebuilding the file. Another sophisticated virus detects whether a compiler is present on the host computer system 200. If a compiler is present, the virus uses it to recompile a slightly modified version of the virus's source code and thereby generate a completely new viral body. Then, the virus inserts the new viral body into the host file. Other metamorphic viruses can use techniques in addition to, or instead of, the ones described herein to modify the viral bodies and insert the bodies into host files.

[0042] FIG. 4 is a high-level block diagram illustrating modules of the VDS 400 for detecting the presence of a virus in a host file or elsewhere in the computer system 200

according to an embodiment of the present invention. FIG. 4 illustrates only the modules of the VDS 400 useful for describing the present invention, and it will be understood that an embodiment of the present invention may include other modules not described herein. In addition, embodiments of the present invention may lack modules described herein and/or distribute the described functionality among the modules in a manner different than described herein.

[0043] The VDS 400 includes a P-code data module 410, an engine module 412, an emulating module 414, and a virus database module 430. The P-code data module 410 preferably holds P-code instruction modules for controlling the operation of the VDS 400 to detect the presence of a virus. As used herein, "P-code" refers to program code for providing data-driven functionality to the VDS 400. Preferably, a virus researcher creates the P-code instructions in the data module 410 by writing instructions in any computer language and then compiling the instructions into P-code. In addition, the functionality of the P-code can be replaced by other data-driven techniques. For example, the program code can be stored in a representation other than P-code or a state machine can be utilized in combination with, or as an alternative to, the P-code in order to provide deterministic data-driven virus detection.

[0044] A preferred embodiment of the present invention uses data-driven techniques to control the operation of the VDS 400 because such techniques allow the functionality of the VDS to be updated by updating the P-code and/or other data. For example, the VDS 400 can be updated to detect new viruses by updating the P-code instead of the other modules. Accordingly, the use of data-driven techniques simplifies the task of updating thousands or millions of VDSs 400 that are installed on computer systems "in the field." However, alternate embodiments of the present invention implement the described functionality through non-data-driven techniques.

[0045] The engine module 412 preferably controls the operation of the VDS 400 in response to the P-code in the P-code data module 410. The engine 412 preferably contains a P-code interpreter 416 for interpreting the P-code, which in turn controls the operation of the engine 412. In alternative embodiments where the data module 410 holds instructions in a format other than P-code, the engine 414 is equipped with a module for interpreting or compiling the instructions in the relevant format.

[0046] The P-code interpreter 416 preferably includes special P-code function calls called "primitives" 418. The primitives 418 can be, for example, written in P-code or a native language, and/or integrated into the interpreter 416 itself. Primitives 418 are essentially functions useful for examining the host file and virtual machine 422 that can be called by other P-code. For example, the primitives 418 perform functions such as opening files for reading, closing files, zeroing out memory locations, truncating memory locations, locating exports in the file, determining the type of the file, and finding the offset of the start of a function. The functions performed by the primitives 418 can vary depending upon the computer or operating system in which the VDS 400 is being used. For example, different primitives may be utilized in a computer system running the MACINTOSH operating system than in a computer system running a version of the WINDOWS operating system. In an alternative embodiment, some or all of the primitives can be stored in the P-code data module 410 instead of the interpreter 416.

[0047] The emulating module 414 is preferably adapted to execute computer program instructions in the host file in a virtual machine under the direction of the engine module 412. The emulating module 414 includes a control program (CP) module 420 for setting up a virtual machine 422 having a virtual processor 424, a virtual memory 426, and a set of virtual registers 428 (typically part of the virtual processor). The virtual machine 422 can emulate a 32-bit MICROSOFT WINDOWS environment, an APPLE MACINTOSH environment, or any

other hardware and/or software environment for which emulation is desired. The virtual machine 422 uses the virtual processor 424 to execute the instructions in the virtual memory 426 in isolation from the actual hardware and software on the computer system 200. Thus, a virus or other malicious code executed by the emulating module 414 cannot contaminate the computer system 200.

[0048] The virus database module 430 preferably stores data utilized by the VDS 400 to determine whether a file is infected by a virus. In one embodiment, the virus database module 430 stores data describing the known viruses that can be detected by the VDS. For each virus, the database module 430 preferably stores data 432 describing the infection characteristics of the virus, data 434 describing how to detect the presence of the virus, and data 436 describing how to repair a file infected with the virus (if possible). In one embodiment, the virus database module 430 also stores other data, such as data representing characteristics that can be utilized to detect unknown viruses.

[0049] In one embodiment, the infection characteristics 432 stored by the virus database module 430 include the markers left by viruses to indicate that a virus has infected a file. Typically, viruses leave such markers in a file to avoid re-infecting the same file. However, the markers may also occur naturally and, therefore, cannot be solely relied upon to indicate the presence of a virus. The infection characteristics 432 also describe the types of files infected by the viruses, how the viruses infect the files, etc. These characteristics are preferably utilized by the file selection module 510, described in more detail below, to determine whether a file potentially hosts a virus.

[0050] In a preferred embodiment, the data 434 in the virus database module 430 describe how to detect the presence of a virus based on register values. As described above, metamorphic viruses typically do not have fixed sequences of instructions that can be utilized

to detect the virus. Still, metamorphic viruses typically achieve instruction polymorphism by replacing instructions with one or more different instructions for performing the same functions. Therefore, a given segment of a metamorphic virus (e.g., a function, routine, or module in the virus) will act on a fixed set of inputs to produce a fixed set of outputs.

[0051] Consider a metamorphic virus that has the following instructions in a first generation:

```
MOV BP, 9090
MOV AX, 9192
XOR BX, BX
PUSH AX
```

The first two instructions place the values 9090 and 9192 in registers BP and AX, respectively. The third instruction sets the value of the BX register to zero, and the fourth instruction pushes the value 9192 onto the stack.

[0052] Assume that a second generation of this virus changes the instructions performing these functions to:

```
MOV BP, B0A0
SUB BP, B0A2
ADD BP, 9194
SUB BP, 102
MOV AX, 9092
ADD AX, 100
XOR BX, BX
PUSH AX
```

These instructions are quite different from the instructions in the first generation of the virus. Nevertheless, these instructions accomplish the exact same tasks as the first generation instructions, namely placing the value 9090 in register BP, zeroing the value of register BX, and pushing the value 9192 on the stack from register AX.

[0053] Although both generations of the virus in this example use the same registers, other viruses may change the registers used by different generations. However, all generations of the

same virus will typically store the same values in the same order if given the same set of inputs. Accordingly, the values the virus stores in the registers, and the order in which the virus stores the values, can be utilized to detect the presence of the virus. For purposes of this description, the set of values placed in the registers by a virus, and/or the order that the values are placed in the registers, are referred to as the virus's "register signature." The register signature is preferably developed by virus researchers who analyze the functional behavior of the virus and identifies a signature that is highly-indicative of the presence of the virus.

[0054] The register signature differs from the traditional "signature" utilized by traditional anti-virus software because the register signature identifies outputs produced by the viral code, while the traditional signature identifies the viral code itself. In a preferred embodiment of the present invention, however, a register signature can also specify one or more opcodes that may be present in the viral code or otherwise executed by the virus. Allowing opcodes in the register signatures enhances the present invention's ability to detect viruses because it gives virus researches greater flexibility in fine-tuning the register signatures to identify particular viruses.

[0055] The virus database module 430 preferably represents a register signature for a virus as a list of one or more predicates that the virus is expected to satisfy. The entries in the list are preferably in sequential order, meaning that the virus is expected to satisfy an earlier-listed predicate before satisfying a later predicate. For example, in one embodiment the register signature for the sample virus described above is as follows:

```
[REGISTER_SIGNATURE_BEGIN]
        regBP = 9090
        regAX = 9192
        regBX=0
[REGISTER_SIGNATURE_END]
```

In this example, the labels "[REGISTER_SIGNATURE_BEGIN]" and "[REGISTER_SIGNATURE_END]" respectively indicate the beginning and end of the signature. The entries in between the labels each contain a predicate. In this example, each predicate specifies a particular register and a value that the register must contain in order to satisfy the predicate. The registers in this example are BP, AX, and BX and the respective values are 9090, 9192, and 0. Thus, this register signature matches the sample virus described above.

[0056] Since metamorphic viruses often change the registers used by the instructions, a preferred embodiment of the present invention allows predicates to specify registers with wildcards. In addition, the preferred embodiment also allows a predicate to specify multiple equivalent register values. For example, consider the following register signature:

```
[REGISTER_SIGNATURE_BEGIN]
        regBP = 9090, regSP=9090
        regAX = 9192, regBX=1234
        regBX=0, regCX=0
        reg?? = 5678
[REGISTER_SIGNATURE_END]
```

In this signature, the first predicate is satisfied if register BP or register SP contains the value 9090. The second predicate is satisfied if register AX contains the value 9192 or if register BX contains the value 1234. The third predicate is satisfied if either register BX or register CX contains the value of zero. The last predicate specifies a register using a wildcard, "??," and is satisfied if any register contains the value 5678.

[0057] In one embodiment, a register signature can use wildcards and logical expressions to specify the values stored in the registers. For example, a predicate can state "regBX != 0," meaning that the predicate is satisfied if the value of register BX is not zero, or "regBX = 0 & regCX = 1234," meaning that the predicate is satisfied if the value of register BX is zero and the value of register CX is 1234. Similarly, a predicate can state "regCX = F??A," meaning

that the predicate is satisfied if register CX contains a value beginning with "F" and ending with "A." A register signature can also state that a predicate is satisfied if one of multiple values are in a register. For example, the predicate "regBX = 0011, A00D" is satisfied if register BX contains the value 0011 or the value A00D. Further variations of predicates in register signatures will be apparent to those of skill in the art.

[0058] In addition, the present invention preferably allows register signatures to specify opcodes in the same manner that the signatures specify registers. For example, a register signature can state "OPCODE = 8B1A," meaning that the predicate is satisfied if the opcode executed by the processor is 8B1A. Since register signatures specify opcodes and register values in the same manner, the terms "register" and "virtual register" used herein also refer to opcodes whenever appropriate.

[0059] The virus database 430 preferably contains register signatures and additional data specifying how to use the register signatures to detect viruses in files. In one embodiment, these data include emulation parameters specifying how to emulate the computer program instructions in order to make viruses apparent. Depending upon the embodiment and/or virus, these parameters may specify the initial state of the virtual machine 422, the start point or points in the host file from which to begin emulation, the number of instructions to emulate, how to respond to a breakpoint or exception, virtual registers to track for correspondence with register signatures, etc. Some or all of this information can be specified as default values. In one embodiment, the default initial state of the virtual machine 422 is for all of the virtual registers 428 to hold zeroes.

[0060] The data 436 in the virus database module 430 describing how to repair infected files are typically highly-specific to the type of virus. Preferably, these data are developed by virus researchers who study the viruses and identify ways to return infected files and computer

systems to their original states. Some viruses damage files and/or computer systems beyond repair, and the data in the database module 430 preferably indicate if this is the case.

[0061] FIG. 5 is a high-level block diagram illustrating a more detailed view of the P-code data module 410. Embodiments of the P-code data module 410 may contain additional or different modules than those described with respect to FIG. 5 and the functionality may be distributed among the modules in a different manner than is described herein.

[0062] A file selection module 510 preferably contains P-code for selecting a potential host file on the computer system 200 to examine for the presence of a virus. In general, the file selection module 510 filters the files on the computer system 200 to select only files that are susceptible to infection by a virus. In one embodiment, the file selection module 510 performs one or more tests on each file, and only those files that pass each test are designated "susceptible to infection by a virus." The tests can include, for example, determining whether the file contains executable instructions (e.g., whether the file has a ".EXE" or ".DLL" extension), whether the file is a PE file, whether the file contains a marker that is consistent with infection by a known virus, whether the size of the file is suspicious, whether the internal characteristics of the file indicate a possible viral infection, etc.

[0063] In one embodiment, other modules in the VDS 400 are adapted to identify files on the computer system 200 and utilize the file selection module 510 to determine whether the file is susceptible to infection. For example, the other modules may be configured to detect when a file on the storage device 208 is accessed and then activate the file selection module 510 to process the file. In another example, the other modules may be configured to recursively search for files on the storage device 208 and utilize the file selection module 510 to examine each encountered file. In another embodiment, the functionality described in these two examples is provided by the file selection module 510 itself.

[0064] The P-code data module 410 preferably includes an emulation control module 520. This module 520 preferably contains code for identifying and selecting potential virus entry points in the file and emulating instructions in the file at each selected point. Although a preferred embodiment of the emulation control module 520 selects only certain locations in the file as potential entry points, one embodiment of the module treats every instruction in the file, or every instruction within certain regions of the file, as potential entry points. The emulation control module 520 accesses the virus database module 430 to determine the parameters for the emulations and interacts with the control program 420 in the emulating module 414 in order to perform the emulations according to the parameters. In one embodiment, the emulation control module 520 holds the emulation parameters instead of the virus database 430.

[0065] The P-code data module 410 also preferably includes a table builder module 522. This module 522 preferably contains code for building a table tracking the state of the virtual registers 428 and the opcodes during a given emulation of instructions. In a preferred embodiment, the table tracks only a subset of the virtual registers and the opcodes. The subset of registers to track is preferably specified by data stored in the virus database 430, although other embodiments of the present invention may use other techniques to specify the registers. The VDS 400 preferably stores the table in the emulation module 414 or another location in the computer system 200 where the table is accessible to the modules in the VDS.

[0066] FIG. 6 illustrates a register table 600 according to an exemplary embodiment of the present invention. Preferably, the table 600 has a column 612 counting the emulation iterations (i.e., counting the number of instructions emulated by the virtual machine 422). In one embodiment, this column 612 simply maintains a count of the emulated instructions. In

another embodiment, the column 612 tracks the value of the virtual instruction pointer register at each step of the emulation.

[0067] The table 600 also preferably includes one to N columns 614 tracking the values in the specified virtual registers 428 at each step of the emulation. The first row 618 of the illustrated table 600 shows values of zero in all of the virtual registers because a preferred embodiment of the present invention starts the emulation with the registers in this state. The instructions emulated by the virtual machine 422 cause the registers to hold different values in subsequent iterations. Another column 616 in the table 400 specifies the opcode of the instruction executed by the virtual machine 422 at each iteration.

[0068] Although the virtual registers in the illustrated table 600 hold 32 bits, in real-world embodiments the amount of data held by the entries in each column can vary depending upon the data being tracked by the column. For example, the entries in columns associated with particular virtual registers preferably hold amounts of data equal to the widths of the associated registers. The values illustrated in the entries the table 600 of FIG. 6 are merely examples, and are not representative of a particular emulation of the virtual machine 422.

[0069] The overall size of the table 600 depends upon the particular embodiment of the present invention. In one embodiment, the total size of the table is less than 64K. In one embodiment, if the number of iterations exceeds the number of rows in the table, the table builder module 522 wraps back to the start of the table 600 and continues to fill the table entries.

[0070] Returning to FIG. 5, the P-code data module 410 also preferably includes a breakpoint condition module 524. This module 524 preferably specifies breakpoints and other

stopping conditions for the emulation. Preferably, the emulation control module 520 utilizes the breakpoint condition module 524 to stop the emulation at certain points.

[0071] In one embodiment, the breakpoint condition module 524 specifies a number of instructions that should be emulated before stopping. Preferably, the breakpoints are configured to stop emulation after relatively few instructions. For example, in one embodiment a breakpoint stops emulation after 50-500 instructions. In contrast, typical emulation-based virus detection software may emulate over 1 million instructions before reaching a breakpoint.

[0072] In another embodiment, the breakpoint condition module 524 specifies a virtual machine state at which emulation should stop. In still another embodiment, the module 524 specifies a combination of factors for determining whether to stop emulation. For example, the breakpoint condition module 524 can specify that emulation should stop if more than 75 instructions were emulated and the BP register holds the value "9090." In another example, the breakpoint condition module 524 can specify that a breakpoint should occur every 100 instructions, until the total number of emulated instructions exceeds a specified maximum.

[0073] In one embodiment, the breakpoint condition module 524 stores separate breakpoint conditions for certain host files and/or viruses. For example, in one embodiment the module 524 stores breakpoint conditions to be utilized when the file is in the standard MICROSOFT WINDOWS distribution. Similarly, in one embodiment the module 524 stores breakpoint conditions to be utilized when the VDS 400 is checking for the presence of a particular virus. In one embodiment, the breakpoint condition module 524 also stores default breakpoints for use when there are no specific breakpoint conditions for a given file and/or virus.

[0074] The P-code data module 410 also preferably includes a virus reporting module 526 storing code for detecting the presence of a virus in the file. Preferably, the virus reporting module 526 analyzes the table 600 created by the table builder module 522 to determine whether it contains any of the register signatures stored in the virus database module 430. In one embodiment, the table 600 contains a register signature if the virus reporting module 526 finds a starting point in the table from which every predicate in the register signature is satisfied.

[0075] If the virus reporting module 526 matches a register signature with the data in the table 600, the module preferably reports a probable virus detection to the VDS 400. In response, other modules in the VDS 400 preferably perform actions such as notifying the computer user of the virus, quarantining the file, and attempting to repair the infected file and/or computer system. If the virus reporting module 526 does not match a register signature with the data in the table 600, the module preferably reports this negative result to the VDS 400.

[0076] FIG. 7 is a flowchart illustrating the operation of the VDS 400 to detect the presence of a virus according to an embodiment of the present invention. Preferably, P-code for enabling the VDS 400 to operate in the described manner is stored in the P-code data module 410. Since the operation is data-driven, other embodiments of the present invention may perform different or additional steps than those described herein. In addition, the steps may be performed in different orders.

[0077] The VDS 400 examines the files on the computer system 200 to identify those files that are susceptible to infection by a virus. Preferably, the VDS 400 selects 710 a single file for further examination. Alternatively, the VDS 400 receives a reference to a file from another module executing on the computer system 200 and subjects this file to further examination.

[0078] The VDS 400 examines the selected file in order to identify 712 potential entry points for a virus. As described above, there are many locations in the file that viruses can exploit as entry points. The VDS 400 selects one of the entry points and emulates 714 the instructions at that entry point according to the specified emulation parameters. The VDS 400 also builds 716 a register table tracking the contents of certain virtual registers 428 as the instructions are emulated. The VDS 400 continues to emulate the instructions and build the register table until the emulation reaches a breakpoint. The process of emulating the instructions in the file and populating the register table is referred to herein as "tracing" or "micro-tracing."

[0079] Then, the VDS 400 preferably analyzes 718 the register table to determine if it contains a register signature indicative of a virus. In one embodiment, the VDS 400 compares all of the register signatures in the virus database 430 against the register table. In another embodiment, the VDS 400 compares only a subset of the register signatures against the register table, depending upon variables such as the file being emulated, the entry point, the breakpoint, etc.

[0080] If 720 the VDS 400 detects a virus, it preferably responds 722 to the virus by notifying the computer user, cleaning the file, etc. Depending upon the breakpoint, the VDS 400 may resume the micro-trace of the entry point if it does not detect a virus (this step is not shown in FIG. 7). For example, the VDS will resume the micro-trace if it has emulated only 100 instructions and the breakpoint specifies that a break should occur every 100 instructions until a maximum of 1000 instructions are emulated.

[0081] If the VDS 400 finishes micro-tracing the entry point and it does not detect a virus, the VDS determines 722 whether there are more potential entry points in the selected file to micro-trace. If so, the VDS 400 begins a new micro-trace at the next potential entry point.

When the VDS 400 has micro-traced all of the potential entry points without detecting a virus, it determines 724 whether there are more files to check for viruses. If there are more files, the VDS 400 selects another file and repeats the virus detection process described above. If there are no more files, then the VDS 400 completes 726 operation.

[0082] In sum, a program executed with a given set of inputs will produce a given set of outputs. The present invention uses this property to detect the presence of metamorphic viruses by micro-tracing portions of potential virus host files and comparing the resulting register states with register signatures of known metamorphic viruses. As a result, the present invention can detect viruses that may be undetectable with other means, such as pattern matching.

[0083] The above description is included to illustrate the operation of the preferred embodiments and is not meant to limit the scope of the invention. The scope of the invention is to be limited only by the following claims. From the above discussion, many variations will be apparent to one skilled in the relevant art that would yet be encompassed by the spirit and scope of the invention.

[0084] We claim:

CLAIMS

1. A method for detecting a computer virus in a file on a computer system, comprising the steps of:

identifying a potential entry point for the virus in the file;

emulating instructions in the file at the entry point in a virtual machine having virtual registers;

tracking contents of the virtual registers; and

detecting a computer virus responsive to the contents of the virtual registers.

2. The method of claim 1, wherein there are a plurality of potential entry points for the virus in the file and wherein the emulating, tracking, and detecting steps are performed for at least some of the plurality of potential entry points.

3. The method of claim 1, wherein the emulating step comprises the substeps of:

determining emulation parameters specifying how to emulate the instructions at the entry point; and

emulating the instructions responsive to the determined emulation parameters.

4. The method of claim 3, wherein the emulation parameters specify an initial start state for the virtual registers.

5. The method of claim 3, wherein the emulation parameters specify breakpoints for the emulation.

6. The method of claim 3, wherein the emulation parameters specify virtual registers for the tracking step to track.

7. The method of claim 1, wherein the tracking step tracks the contents of a subset of the virtual registers.

8. The method of claim 1, wherein the tracking step tracks the opcodes emulated by the virtual machine.

9. The method of claim 1, wherein the tracking step comprises the substep of:

building a table having columns for the virtual registers being tracked and rows specifying states of the virtual registers being tracked.

10. The method of claim 9, wherein each row of the table specifies states of the virtual registers being tracked at a step of the emulation.

11. The method of claim 9, wherein the tracking step further comprises the substep of:

populating the table with the states of the virtual registers being tracked during the emulating.

12. The method of claim 1, wherein the detecting step comprises the step of:

determining whether the contents of the virtual registers match a register signature for a computer virus.

13. The method of claim 12, wherein the register signature specifies a sequence of register states that are indicative of a particular virus.

14. A virus detection system for detecting the presence of a computer virus in a computer system, comprising:

a virus database for holding register signatures describing computer viruses;

an emulating module for emulating computer program instructions in a virtual computer system having virtual registers; and

an engine module for analyzing contents of the virtual registers to detect a match with a register signature in the virus database.

15. The virus detection system of claim 14, wherein a register signature comprises at least one predicate that is satisfied when a virtual register contains a specified value.

16. The virus detection system of claim 15, wherein the predicate uses a wildcard to specify the value of the virtual register.

17. The virus detection system of claim 15, wherein the predicate uses a wildcard to specify the virtual register.

18. The virus detection system of claim 14, wherein a register signature specifies a sequence of register values that are indicative of a computer virus.

19. The virus detection system of claim 18, wherein the register signature specifies at least one opcode emulated by the emulating module that is indicative of a computer virus.

20. The virus detection system of claim 14, wherein the virus database is further adapted to hold data describing infection characteristics of a plurality of viruses.

21. The virus detection system of claim 14, wherein the virus database is further adapted to hold data describing how to repair a file infected with one or more of a plurality of viruses.

22. The virus detection system of claim 14, wherein the emulating module emulates the instructions with an initial state and wherein the initial state is determined responsive to data in the virus database.

23. The virus detection system of claim 14, further comprising:

a breakpoint condition module for specifying breakpoint conditions, wherein the emulating module is adapted to emulate the computer program instructions until reaching a breakpoint condition.

24. The virus detection system of claim 23, wherein the computer program instructions are in a file and wherein the emulating module performs micro-traces of the instructions in the file responsive to the breakpoint conditions.

25. The virus detection system of claim 14, wherein the computer program instructions are in a file, wherein the file includes a plurality of potential virus entry points, and wherein the emulating module is adapted to emulate instructions at at least some of the plurality of potential virus entry points.

26. The virus detection system of claim 25, wherein the emulating module is adapted to separately emulate instructions at the potential virus entry points and wherein the engine module is adapted to analyze contents of the virtual registers for each emulation to detect a match with a register signature in the virus database.

27. The virus detection system of claim 25, wherein the emulating module performs a micro-trace of the instructions at at least some of the plurality of potential virus entry points.

28. The virus detection system of claim 14, further comprising:

a table building module for building a table tracking values of at least a subset of the virtual registers responsive to the emulated instructions.

29. The virus detection system of claim 28, wherein the subset of virtual registers tracked by the table are determined in response to data stored in the virus database.

30. The virus detection system of claim 28, further comprising:

a virus reporting module for analyzing the table to determine whether the values of the virtual registers tracked by the table match a register signature in the virus database.

31. The virus detection system of claim 14, further comprising:

a file selection module for selecting among a plurality of files on the computer system to identify files susceptible to infection by a computer virus.

32. A computer program product comprising:

a computer-readable medium having computer program logic embodied therein for detecting a computer virus in a computer system, the computer program logic comprising:

a virus database module for holding register signatures describing computer viruses;

an emulating module for emulating computer program instructions in a virtual computer system having virtual registers; and

an engine module for analyzing contents of the virtual registers to detect a match with a register signature in the virus database.

33. The computer program product of claim 32, wherein a register signature comprises at least one predicate that is satisfied when a virtual register contains a specified value.

34. The computer program product of claim 33, wherein the predicate uses a wildcard to specify the value of the virtual register.

35. The computer program product of claim 33, wherein the predicate uses a wildcard to specify the virtual register.

36. The computer program product of claim 32, wherein a register signature specifies a sequence of register values that are indicative of a computer virus.

37. The computer program product of claim 36, wherein the register signature specifies at least one opcode emulated by the emulating module that is indicative of a computer virus.

38. The computer program product of claim 32, wherein the virus database module is further adapted to hold data describing infection characteristics of a plurality of computer viruses.

39. The computer program product of claim 32, wherein the virus database module is further adapted to hold data describing how to repair a file infected with one or more of the computer viruses.

40. The computer program product of claim 32, wherein the emulating module emulates the instructions with an initial state and wherein the initial state is determined responsive to data in the virus database.

41. The computer program product of claim 32, further comprising:

a breakpoint condition module for specifying breakpoint conditions, wherein the emulating module is adapted to emulate the instructions until reaching a breakpoint condition.

42. The computer program product of claim 41, wherein the emulating module performs micro-traces of the computer instructions responsive to the breakpoint conditions.

43. The virus detection system of claim 32, wherein the computer program instructions are in a file, wherein the file includes a plurality of potential virus entry points, and wherein emulating module is adapted to emulate instructions at at least some of the plurality of potential virus entry points.

44. The computer program product of claim 43, wherein the emulating module is adapted to separately emulate instructions at the potential virus entry points and wherein the engine module is adapted to analyze contents of the virtual registers for each emulation to detect a match with a register signature in the virus database.

45. The computer program product of claim 43, wherein the emulating module performs a micro-trace of the instructions at at least some of the plurality of potential virus entry points.

46. The computer program product of claim 32, further comprising:

a table building module for building a table tracking values of at least a subset of the virtual registers responsive to the emulated instructions.

47. The computer program product of claim 46, wherein the subset of virtual registers tracked by the table are determined in response to data stored in the virus database.

48. The computer program product of claim 46, further comprising:

a virus reporting module for analyzing the table to determine whether the values of the virtual registers tracked by the table match a register signature in the virus database.

49. The computer program product of claim 32, wherein the data module comprises:

a virus reporting module for determining whether the values of the virtual registers match a register signature in the virus database.

50. The computer program product of claim 32, further comprising:

a file selection module for selecting among a plurality of files on the computer system to identify files susceptible to infection by a computer virus, wherein the emulation module emulates computer program instructions in the selected file.

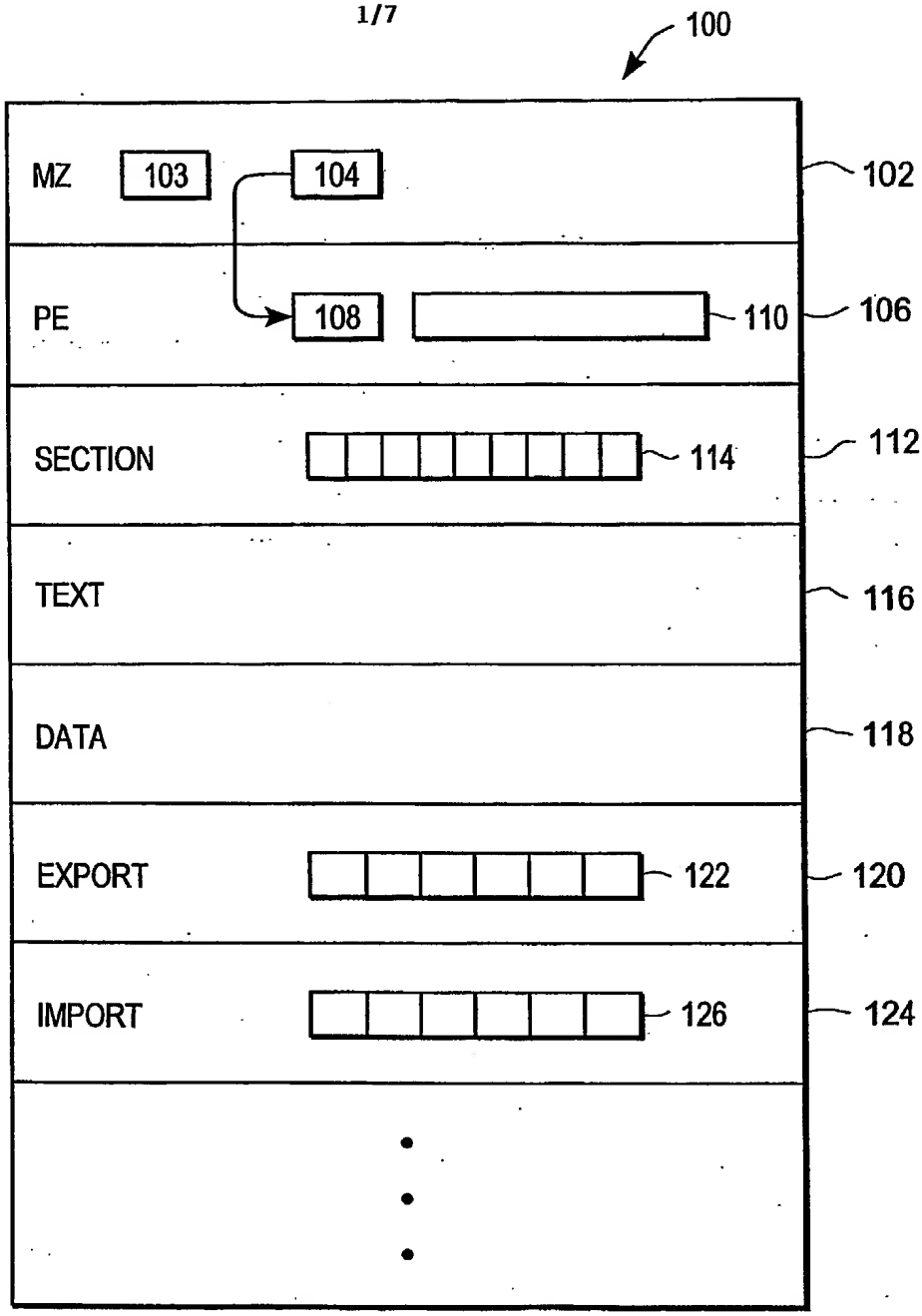

100
MZ
103
104
102
PE
108
110
106
SECTION
114
112
TEXT
116
DATA
118
EXPORT
122
120
IMPORT
126
124

FIG. 1

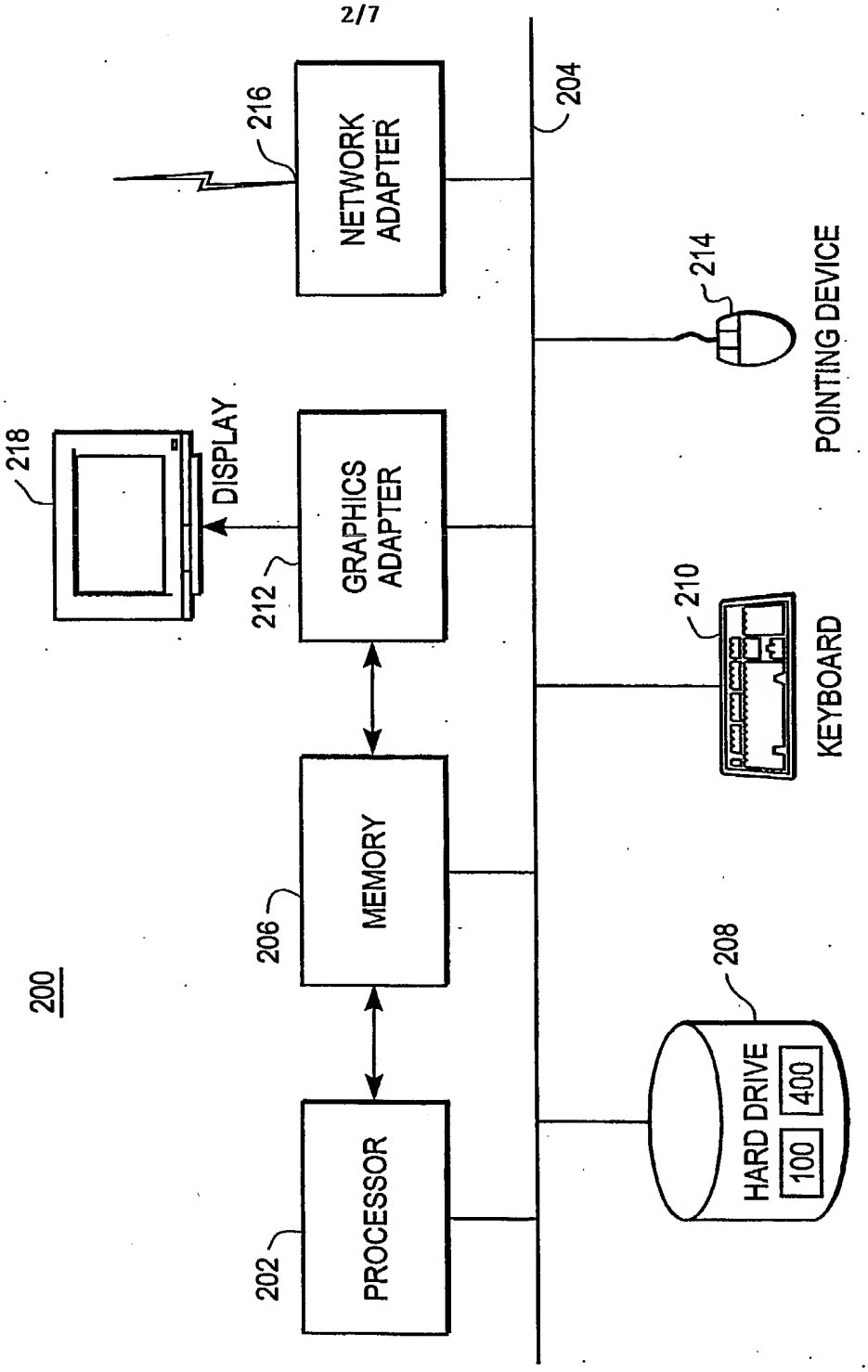

200
PROCESSOR
202
MEMORY
206
GRAPHICS ADAPTER
212
DISPLAY
218
NETWORK ADAPTER
216
204
214
POINTING DEVICE
210
KEYBOARD
208
HARD DRIVE
100
400

FIG. 2

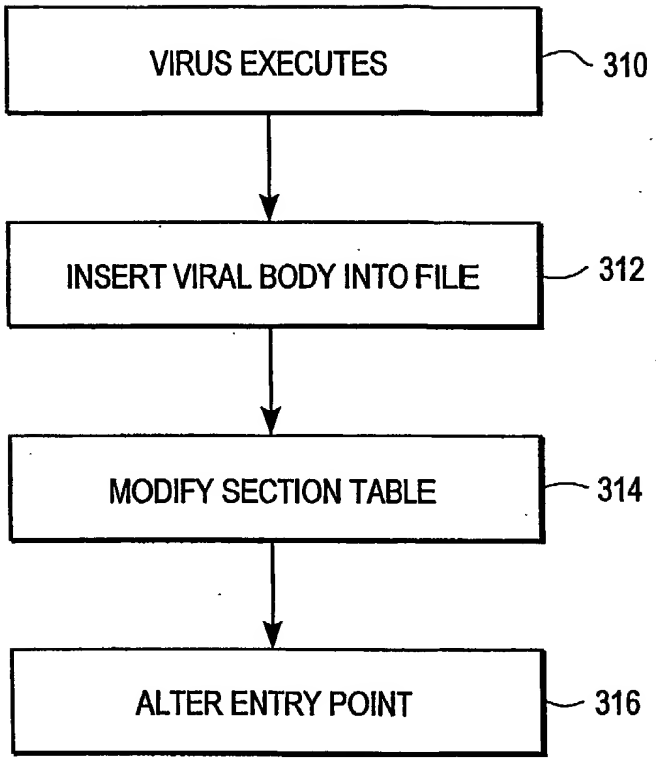

VIRUS EXECUTES
310
INSERT VIRAL BODY INTO FILE
312
MODIFY SECTION TABLE
314
ALTER ENTRY POINT
316

FIG. 3

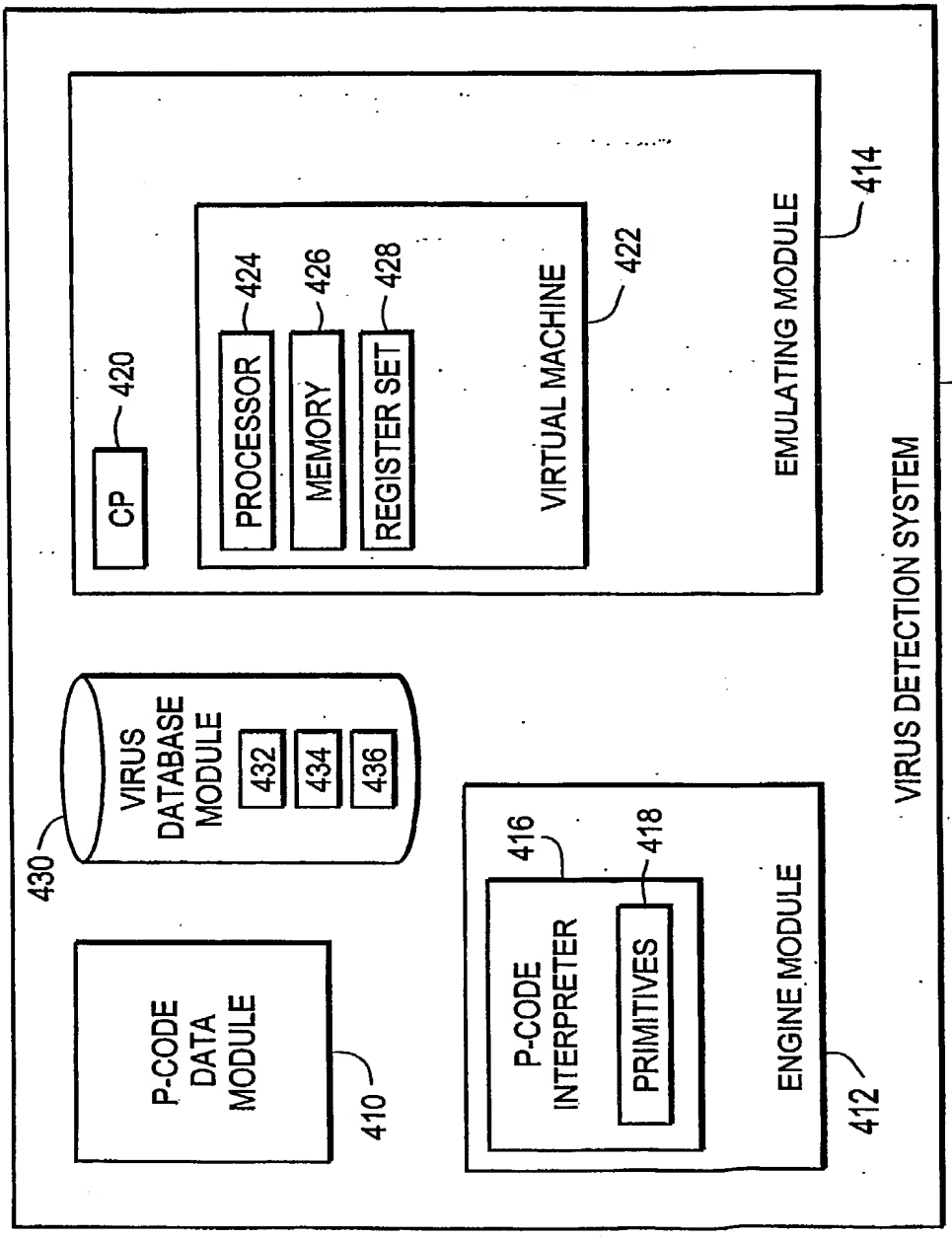

VIRUS DETECTION SYSTEM
400
EMULATING MODULE
414
VIRTUAL MACHINE
422
PROCESSOR
424
MEMORY
426
REGISTER SET
428
CP
420
ENGINE MODULE
412
P-CODE INTERPRETER
416
PRIMITIVES
418
VIRUS DATABASE MODULE
430
432
434
436
P-CODE DATA MODULE
410

FIG. 4

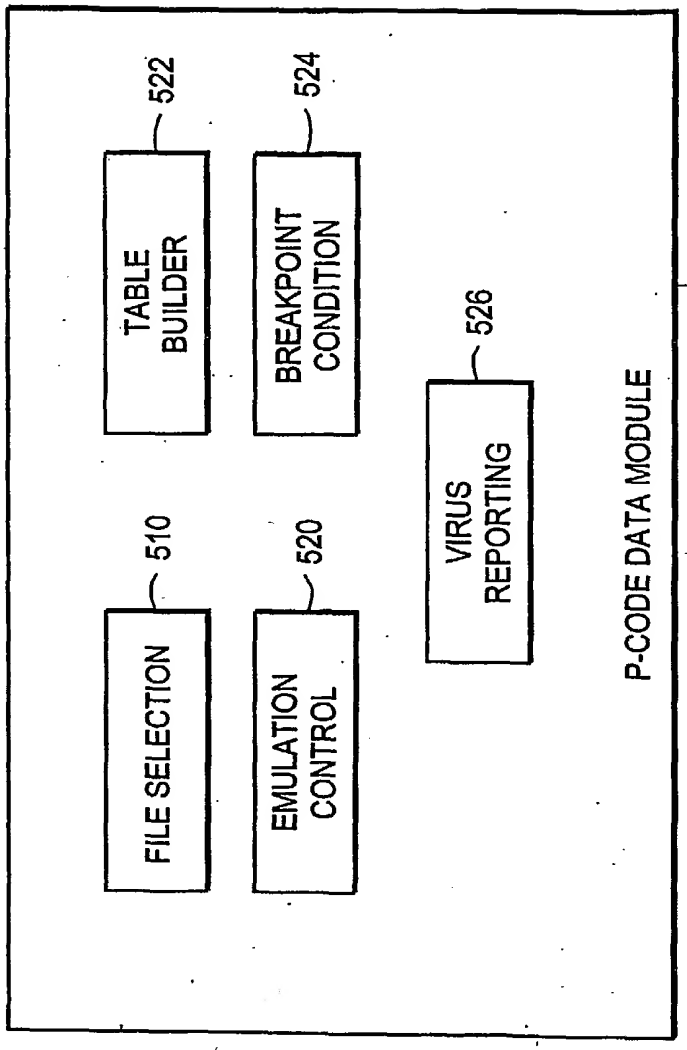

P-CODE DATA MODULE
410
TABLE BUILDER
522
BREAKPOINT CONDITION
524
VIRUS REPORTING
526
FILE SELECTION
510
EMULATION CONTROL
520

FIG. 5

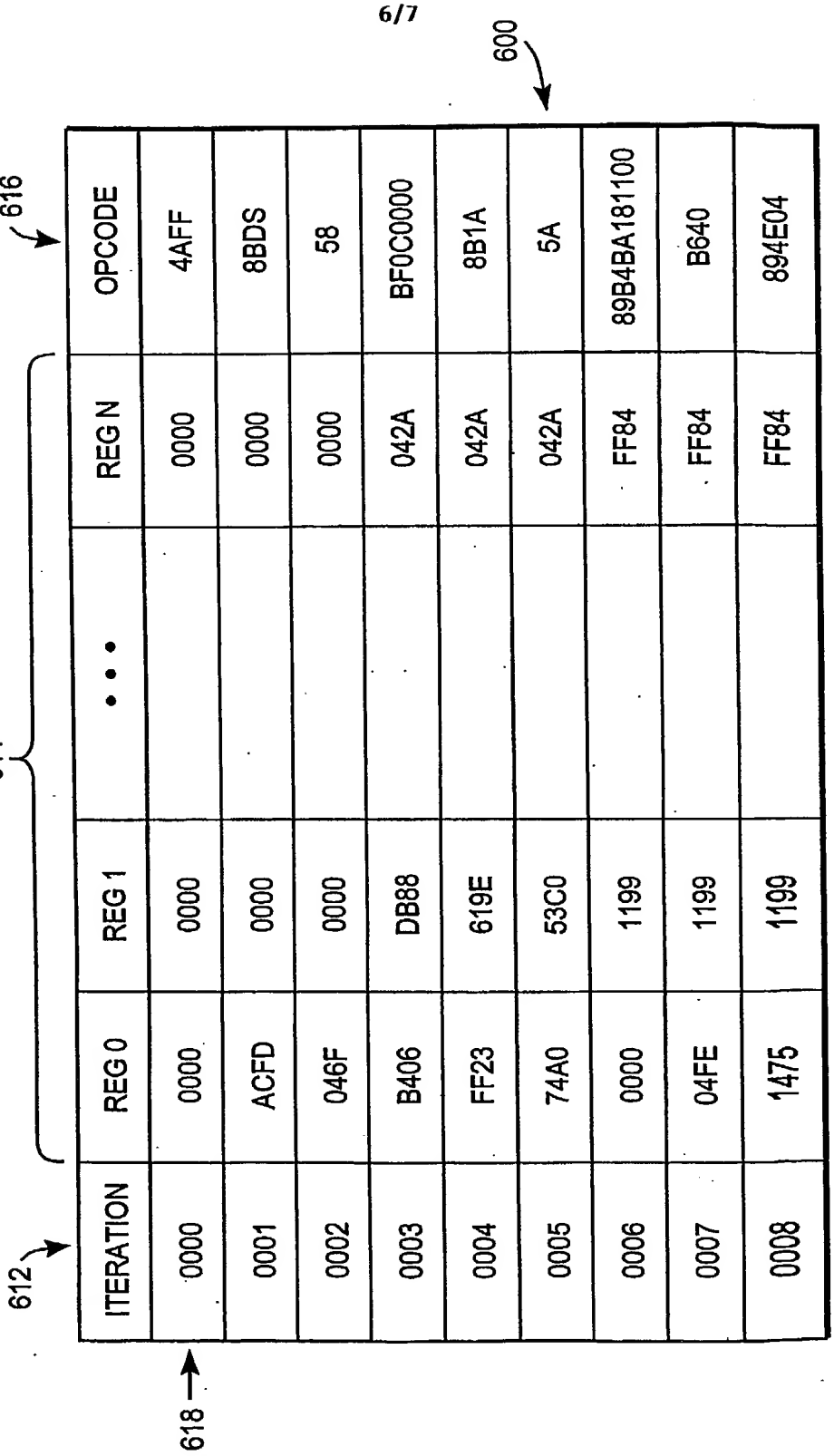

600
612
614
616
618
ITERATION
REG 0
REG 1
• • •
REG N
OPCODE
0000 0000 0000 0000 4AFF
0001 ACFD 0000 0000 8BDS
0002 046F 0000 0000 58
0003 B406 DB88 042A BF0C0000
0004 FF23 619E 042A 8B1A
0005 74A0 53C0 042A 5A
0006 0000 1199 FF84 89B4BA181100
0007 04FE 1199 FF84 B640
0008 1475 1199 FF84 894E04

FIG. 6

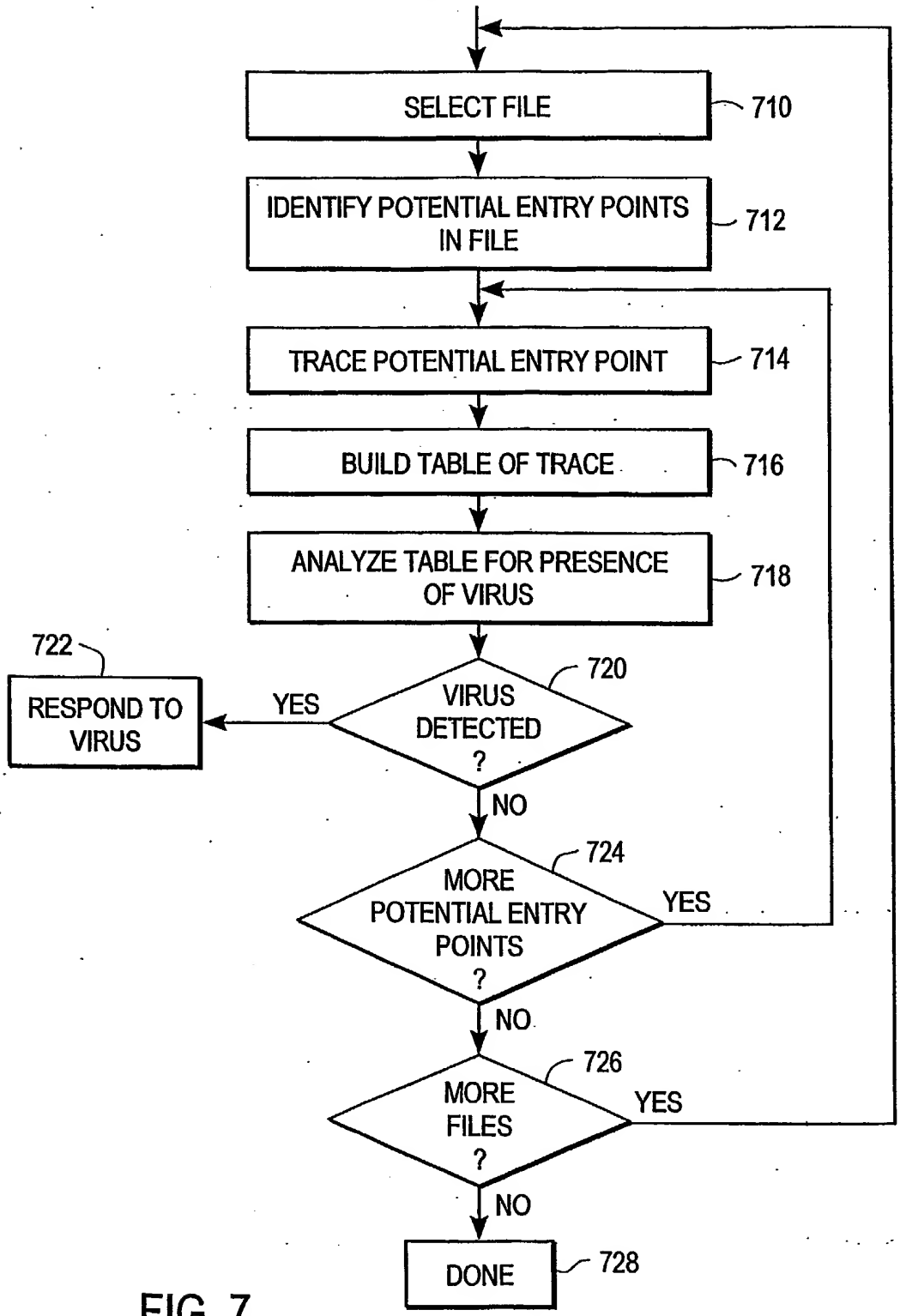

SELECT FILE
710
IDENTIFY POTENTIAL ENTRY POINTS IN FILE
712
TRACE POTENTIAL ENTRY POINT
714
BUILD TABLE OF TRACE
716
ANALYZE TABLE FOR PRESENCE OF VIRUS
718
722
RESPOND TO VIRUS
YES
VIRUS DETECTED ?
720
NO
724
MORE POTENTIAL ENTRY POINTS ?
YES
NO
726
MORE FILES ?
YES
NO
DONE
728

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US03/16445

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : H04L 9/00
US CL : 713/200

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 713/200

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WEST, EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6,067,410 A (NACHENBERG) 23 May 2000, abstract; fig. 1b-4; col. 2, lines 47-67; col. 3, lines 1-67; col. 4-12 and col. 13, lines 1-63	1-50

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 August 2003 (28.08.2003)	10 SEP 2003
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703)305-3230	Authorized officer Gilberto Barron Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)